Computers are a pain to use. How is information stored, arranged and retrieved? How are programs executed? Today's widely used systems answer these questions in ways that are unclear and unintuitive to many people. Users who aren't software developers or computer enthusiasts, who rely on their computers for document-management, communication and canned applications, are least likely to be happy with the current state of things. Microsoft acknowledged that fact when it released a widely-discussed system called "Bob," in which a collection of cartoon characters in a series of rooms guide users through the intricate capabilities of the underlying system. A dog appears, for example, with an attached speech balloon: "Bow-Wow! Add this event... Add this event to which of these calendars?" A list of names follows, and the user checks the right ones.

We might dismiss Bob as a goofy sideshow, but we would be wrong not to take it seriously. "The idea of a social interface as pioneered here will become pervasive" says Microsoft Chairman Bill Gates, who knows a thing or two about pervasiveness. Apple is said to be working on its own Bob-like product for release next year. Pondering Bob yields two inferences. First, the software industry understands that a problem exists. Second, the industry believes that, if you can't figure out how today's operating system environments work all by yourself, you must be a moron and ought to be treated like a child.

We believe on the other hand that people are right to be confused by today's systems; they aren't well suited to the needs of most users. They are built on ideas (like named files in hierarchical directories) that were brilliant when they were new but have long since become obsolete.

We've developed "Lifestreams" in an attempt to do better. The prototype that exists today realizes only some of the system's defining features, but it's up and running and has allowed us to experiment with many of the system's key ideas. In this article we describe the idea, its context, our current implementation and the direction our system is headed.

What is it?

Under the constraints imposed by our current implementation and (more important) by the state of the world generally, a Lifestream as of today is a way of organizing all of a user's dealings with computers, and all his computer-mediated dealings with other people. But the information world is changing fast. Lifestreams are intended ultimately to serve as comprehensive electronic diaries, and to hasten the transition from a paper-centered world to an electronic one. In the electronic world-to-come software handles a good deal of the overhead that comes with managing chunks of information, and paper is reserved

for applications (books for example) where it's a good fit and not merely the only alternative.

A Lifestream, today and always, is a stream of information chunks, typically intended to include *every* information chunk of interest to its owner. Chunks are stored in the stream chronologically by the date and time at which they were created (for chunks created by the owner) or added to the stream (for chunks arriving from the outside world). Users can display portions of the stream arranged in some other way than chronologically, but time is a Lifestream's master-criterion. We designed it this way because time is a natural guide to experience. Memories aren't arranged or recalled by chronology exclusively, and some memories lack a time component altogether; but time is the attribute that seems to come closest to a universal skeleton key to stored experience.

A "chunk" is any chunk of data that would ordinarily be treated as a unit: a document, email message, calendar item, software Rolodex card or fax image, for example, or the transcript or screen image created by an executing application. Lifestreams are also designed to contain chunks that don't ordinarily exist in electronic form today but will in the near future: bills, receipts, business cards, owner's manuals, snapshots, videos, licenses... If you are a student, your transcripts, graded assignments, bulletins and schedules are stored on your stream. If you are a patient, your medical history is stored as a series of discrete chunks (lab reports, prescriptions, doctors' notes). Whoever you are, item number one in your Lifestream-of-the-future is probably your birth certificate.

The system we've built deals only with today's "conventional" electronic chunks. But it's designed to generalize smoothly into a comprehensive electronic diary.

How does it work? (And what's the point?)

The system is based on five observations.

1. "Names" and "directories" should be junked as organizing devices for infochunks. Virtually all current systems require users to make up a new name whenever they create a new document. But names should be required only when users feel like inventing them. When you grab a piece of paper and start writing, no-one demands that you bestow a name on the sheet of paper. Too frequently, computer users find themselves inventing names like "Letter to Schwartz no. 17" or "New Report"; a new document is often merely the latest in a continuing series, and to name it individually is a waste of time. Such names tend not only to be pointless but useless for retrieval purposes.

Directories are one important symptom of a different but wide-ranging problem. In some cases, software fails to match the flexibility of paper systems; more often, software is *too* faithful to the paper world of the past. In most current systems, file directories are a strait-jacket metaphor. Paper can't be in more than one place at a time, but infochunks can be (or can appear to be). Furthermore, it's time-consuming to organize papers into folders and folders into bigger folders, but infochunks can be organized fast. Current systems treat infochunks as if the creation of a folder were a big deal to be approached with care and deliberation. Directories *should* be created on the fly when they are needed, and infochunks should belong to as many of them as seems reasonable, or to none.

Imagine the papers in your office in a single heap (assuming they aren't in a single heap already); we want it to be possible for you to say "hand me a folder holding everything that deals with toaster ovens," and to have the specified folder materialize instantaneously—despite the fact that you may never have organized your chunks by relevance-to-toaster-ovens before, and may never do it again.

2. The system should provide sophisticated logic for finding a chunk or group of chunks. This point is the logical continuation of the previous one. I don't want to be forced to make up pointless names; I *do* want to be able to ask for "my last letter to Pinsky," "the letter to Pinsky about toaster ovens," "everything about toaster ovens," "the report I co-authored with Pinsky" and so on. If I ask for "everything about toaster ovens," ideally the system would be sophisticated enough to include, also, chunks dealing with small appliances, ordinary toasters and so on (but it should indicate that such chunks are less-close fits to my query).

3. The system should provide sophisticated logic for summarizing or compressing (and where appropriate, for picturing or animating) a large group of related chunks of which the user wants a concise overview.

No matter how many chunks fall into a given category, the system should be capable of summarizing the whole lot on a single screen. For some types of chunk, pictures or animations will be good vehicles for summaries.

4. "Compatibility" should be automatic. Computers are ordinarily understood to be devices for storing and managing information. We prefer a different metaphor: a computer is a viewing device for tuning in Lifestreams—and other types of information landscape as well; but your LifestreamsΦ is home base. In this view of the world, the character of a computer and its native software are irrelevant to the computer user. You can tune in your Lifestream from a Unix machine at work, a Mac at home or a PC at the supermarket or in a phone booth. Laptops become unusual devices for special tastes and circumstances; you won't ordinarily carry a computer when you travel, any more than you'd

carry a TV today. It's safe to assume that, en route, you'll find a TV in your hotel room that will be capable of tuning in exactly the same things your own personal TV could have managed. So you might just as well leave your own personal TV home. Ditto the computer, once the Age of Lifestreams is underway. In the Lifestreams world, you find computers installed in most places where you might want to deal with your infochunks, and you grab whatever unit is free.

5. Explicit file-storage management should be junked. Longer-term storage management is another area in which current systems fail to deliver the power of which software is capable. I own a collection of paper documents, and I might be willing to pay someone to make sure that they are stored safely and that I'll be able to find each one quickly when I look for it; but unless I rent a safe-deposit box, there's no convenient way in which a company could provide such a service. It would have to show up regularly at my home and office, copy important papers and truck them off somewhere. I also own a collection of software objects, and once again I might be willing to pay someone to make sure they are stored securely and reliably. The Lifestreams systems is intended to provide such a service. It accepts responsibility for keeping your objects safe and available, and you can throw away your diskettes.

Our current system realizes a fair part of goals 1 and 2, some of 3 and 5 and very little of 4. But all these goal are central to the project.

The Lifestreams System

We've sketched the idea of a Lifestream and the system's goals. Here we describe how the system works, explaining in the process our current prototype and future plans.

The Stream

A lifestream can be accessed from any available "viewport." (A viewport is any computer running the necessary software.) When a user logs on from a viewport, he sees the master chunk list of the stream he's just attached to—the master chunk list being a scrollable list of all chunks on the stream. In our current system, the chunk list takes the form of a text table; each entry describes one chunk in terms of a subset of the chunk's attributes. The most-recent chunk is at the top of the list. As new chunks are created or arrive from the outside, the chunk list is updated.

We are in the process of developing a new viewport that displays the master chunk list as a 3D stream receding from the present into the past. 3D is helpful

because it allows us to use visual cues to communicate important information about chunks—namely, their relative ages; what's old and what's new.

A viewport presents buttons for managing Lifestreams. Two buttons, *new* and *clone*, create new chunks. *Clone* can duplicate a chunk on the stream, or a template from a pre-arranged collection of commonly-used types—for example, email. (A template is actually just a chunk on the stream. Users can add templates, or use any pre-existing chunk as a template.) A third button, *copy*, allows the user to copy a chunk from his stream to someone else's. Sending email represents one use of *copy*—to send email to Ginger, you copy a chunk to Ginger's stream.

The viewport also allows the user to examine a chunk's attributes more closely, and to grant other users' selective access to his chunks. The *find* button creates a substream: it prompts the user to specify a set of attributes or query words and then creates a substream to order, as we discuss below. The viewport can then be focused on the new substream—or it can be tuned to any existing substream.

Attributes and substreams

Each chunk in the stream has a collection of associated attributes. Some are supplied by the system when a chunk is created or copied; others are supplied explicitly by the user. (When you make a copy of a chunk, the copy inherits the original's attributes, except for creation time.) The system uses attributes to locate chunks. If a user feels the urge to name a chunk, he supplies the name as one attribute; the system can then retrieve the chunk under its name. The date and time at which a chunk is created or appended are attributes; a chunk's origin or destination (if it is copied among streams) are attributes. When a chunk follows the outline that is associated with letters (beginning with an address, date and salutation, ending with a closing), the system can infer that "letter to Ginger" should be one of its attributes, where "Ginger" is copied out of the address or salutation. When a chunk consists only of a name and address (or phone or email information), it can infer that "Rolodex card for Ginger" should be an attribute, and so on.

The system has a "find" button whose function (as we have said) is to create a substream. A substream is a list of all chunks that satisfy some search criterion; it may include information on "closeness of fit" if the criterion (e.g. "find everything dealing with toaster ovens") can be "approximately" satisfied. (The system's *find* button is powered by an expert database called the "FGP system" developed at Yale in a separate research project.) Chunks on the substream remain part of the main stream. Ordinarily, the list of substream chunks

is ordered just like the stream itself and its associated list. But under some circumstances the order will be different: it will display the "closest" chunk first, or use some other ordering criterion specified by the user—it might be ordered by size, alphabetically and so on.

A substream lives until it is killed. If you've created a toaster-oven substream and a new chunk chock-full of information on toaster ovens arrives subsequently, it is added to the toaster-oven substream in the process of being appended to the main stream. Finding a unique chunk is a special case of creating a substream. If you've named some chunk "Audrey," then "find Audrey" locates that particular chunk. (Operations like "find my last letter to Schwartz" would also return a unique chunk, as would many others.) Note that attributes and substreams allow users to create traditional hierarchical directory structures if they like. A file that would be named "/usr/local/schwartz/frog" in Unix corresponds to a chunk with the attribute "/usr/local/schwartz/frog." The "find" command can assemble substreams corresponding to particular directories by intelligent scanning of file-structured attributes structured as file names.

"Hide" is the inverse operation: it assembles a substream and then suppresses it. Chunks on a "suppressed substream" remain invisible (they don't appear on the master chunk list) until the hidden substream is dissolved. Suppressed substreams are important, because Lifestreams are all-inclusive electronic diaries and may include all sorts of chunks that a user wouldn't ordinarily refer to. Of course a user may if he chooses remove a chunk from his stream, and thereby throw it out. In the near term, chunk-removal is an operation users will perform regularly. Over the long term, though, cost per bit for long-term storage will be so low (remember, storage facilities will be provided by a local information utility—you won't have stacks of diskettes cluttering up your desk) that chunk-removal will most likely be rare. Even in the case of a chunk that is mere electronic scratch paper, the possibility (however small) that you will have reason to refer to it again, plus the inconvenience (however small) of pushing the "delete" button, will outweigh the negligible cost of storage.

(When a user dies, however, the heirs might neaten up the deceased's stream a bit before forwarding it to a local museum or appending it to their own.)

Past, Present and Future

When a new chunk is created, the system assumes that the user will actively work on it for awhile—although presumably it will soon be superseded at the end of the stream by new chunks (e.g. by arriving email). After a chunk hasn't been worked on for "awhile" ("awhile" is a resettable system parameter), it freezes into "history"—that is, it's marked read-only.

The system treats the past, present and future uniformly, insofar as that makes sense. By default the system clock tells current time. Dialing the clock into the past allows you to explore past regions of the stream. You can dial the clock into the future too. Suppose you've scheduled a meeting for next Tuesday afternoon and want to remind yourself of the fact on Tuesday morning. You can dial the clock to next Tuesday morning and create a chunk to serve as a reminder notice; then you return to the present. Next Tuesday morning the chunk you created will be "received" and you will be alerted in the same way as if this chunk were newly-appended email. By dialing into the future you can examine such reminders as they sit around waiting for their time to come.

Agents

Every chunk in a Lifestream has at least one agent attached. An agent is tied onto each infochunk the way a helium balloon might be tied down to a chunk of plastic—sort of. If the chunk represents a document, its agent is the "helper application" by means of which it is manipulated. You view and edit documents using the applications you are familiar with; if you like emacs, the documents on your stream will all have emacs as their attached helper applications.

To run an application (say "Crazy Space Artichokes") under Lifestreams, you create a new, empty chunk and make Crazy Space Artichokes serve as an agent. (In practice, Crazy Space Artichokes becomes a template, like "email"; clicking on the template creates a new chunk with the application attached.) When you are done with this particular Crazy Artichokes session, you snip the agent loose and it disappears, leaving its final screen image behind as a permanent record in the stream—unless you choose to delete it. The helper agent that serves as your default chunk-viewer (presumably your favorite editor) is attached instead.

A chunk might have two agents attached simultaneously—an "executive assistant," for example, designed to shepherd it through the system from stream to stream, in addition to the editor. If a chunk is to be forwarded to a list of users in sequence, the chunk's originator attaches such an executive assistant agent and hands it the list of stream destinations. ("Chunk with attached executive assistant" is, once again, a template.)

Agents may also be assigned to scan publicly-accessible streams, finding interesting chunks and returning them. Such stream-scanning agents can be used for subscriptions, for example—you can attach a subscription agent to the Lifestream representing some electronic magazine, and have the agent append each month's issue to your stream. (Or you might have the agent check, first, whether the new issue has anything to do with toaster ovens, and forward it only if it does.)

Agents can act together. A "meeting maker" application, for example, combines stream and chunk-based agents. A stream-scanning calendar agent keeps track of each user's availability by monitoring the appointment chunks that accumulate in the "future" part of the Lifestream. When a meeting needs to be scheduled, chunk-based agents are dispatched to a meeting's intended participants, where they interact with the calendar agents and possibly with the participants themselves.

The Implementation

Our current prototype manages Lifestreams through servers running on Unix workstations. Each server stores the documents, attributes and substreams of a number of users. The chunks in a Lifestream are represented using a machine-independent external data representation (XDR), so that a stream can be accessed from many different platforms.

So far we've implemented only one viewport; it runs on Unix workstations and presents an X Windows Motif interface to users (see screen snapshot at the end of this document). (In principle, of course, many different viewports, running on different machines and with different user interfaces, can coexist.)

When you select a chunk from the master chunk list, ordinarily it is displayed in its own window; if you select several chunks, each appears in a separate window. But you can also create a "shared window"—whatever chunk you point to in the master list appears in the shared window. A shared window allows fast switching with minimal fuss, bother and screen redecorating among several contexts—for example, the document you're working on and new email arriving on the end of the stream.

When you use "find" to build a substream, the new substream's chunk list replaces the master list. In general, the user can focus the viewport on whatever substream he chooses, or on the entire stream; the appropriate chunk list is displayed.

We have taken a general approach to handling documents—rather than committing to a fixed document standard, such as Microsoft Word or Adobe PDF, we allow documents to be typed with MIME-types, which makes it possible to support a large range of document types.

A Macintosh-based viewport is under development (it displays the master chunk list as a 3D image, as described above); we plan to develop a PDA-based viewport as well.

Other systems' relations to Lifestreams

Lifestreams overlaps many existing systems in many ways. But the two systems with the closest and most interesting relationship to ours are Lotus Notes and Xerox's Tapestry system.

Lotus Notes provides databases for storing document collections, and allows custom-designed viewing and filtering of particular databases by means of a "view." (A view specifies which documents within a particular database should be displayed, and how.) In Notes as in Lifestreams, email and documents are treated uniformly.

Notes doesn't provide Lifestreams-style nameless documents, nor does it provide Lifestreams-style dynamic substreams, nor is it designed to provide Lifestream-style substream summaries, nor does it have the synchronization characteristics of a Lifestream. These aren't flaws in Notes; Notes and Lifestreams are targeted at different problems. Notes is intended mainly not for individual electronic diaries but for corporate documents and information. Thus, Notes databases are statically configured and database filters ("views") aren't typically created by end users but rather by system administrators.

Another system with significant similarities to Lifestreams is Xerox PARC's Tapestry. Tapestry is intended to manage incoming articles (for example from netnews or mailing lists) for a workgroup. As new articles arrive in Tapestry they are collected and archived within a global database. Users don't access the collection of documents directly; instead, they supply content-based filters which forward to each user documents that might interest him. At the user mailboxes, "appraisers" may sort mail into folders or assign it priorities before presenting the user with a list of new documents. Like Lifestreams, Tapestry can infer mail-related attributes for each document, which can be used in filtering.

Tapestry folders correspond roughly but not in detail to substreams. More important, the two systems are intended for somewhat different niches. Tapestry is intended to be an interface between a user and the world of documents streaming in from outside, not a replacement for a user's own filespace.

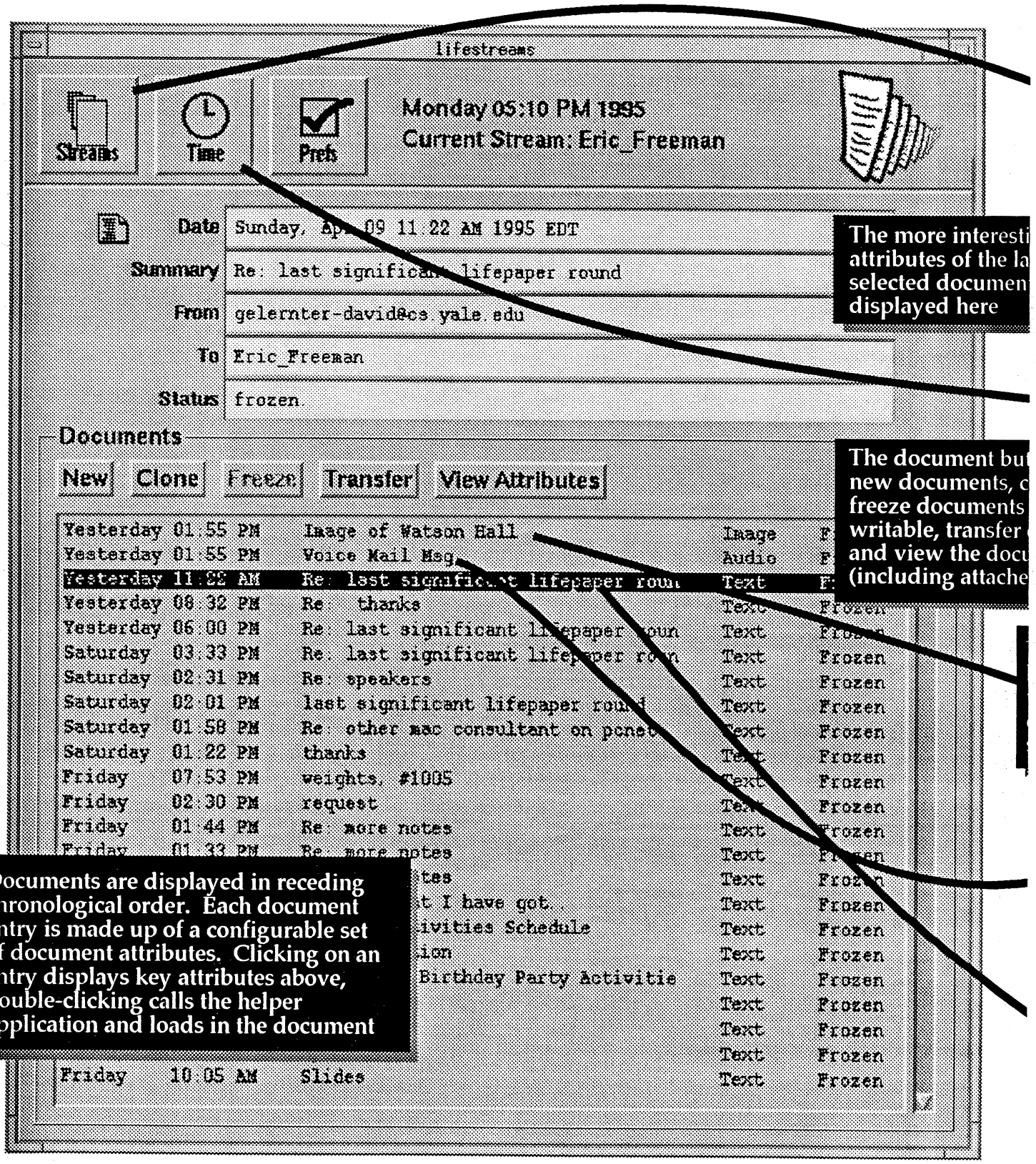
Disk catalogers and indexers are related to Lifestreams in a general way. Either statically or incrementally these programs index the documents of an entire file system, and support efficient searches based on a file's contents. Users of these systems tend to abandon, for the most part, the hierarchical organization of their native file systems in favor of the quick searching and filtering supported by the catalogers. The functionality provided by these programs is subsumed by Lifestreams (and is being incorporated into new versions of conventional operating systems as well).

Many other systems are related to Lifestreams one way or another: meeting makers, document management systems, to-do lists and reminders, enabled-mail (which inspired our instantiation of agents), semantic file systems, archival/ retrieval systems and a large number groupware applications.

Conclusions

Current systems don't begin to exploit the power of modern software and networked computers. In a few years, information management is bound to look dramatically different. We can't tell exactly how it will look, but do know this: the future won't be based on cartoon characters trying desperately to convince us that our current systems make sense.

The modest goal of the Lifestreams project is to change the world, computationally speaking. We have developed utilities that translate pre-existing mail files and Unix file systems into Lifestreams, and provide an interface between Lifestreams users and users of conventional systems. On this basis a number of people at Yale have already migrated to Lifestreams (although the current system is just a prototype and only for the venturesome); more will be joining soon. Lifestreams may or may not develop a large user community eventually. One thing that's almost certain, though, is that the system will develop some sort of user community; and over the near term, that community is likely to grow.


lifestreams
Streams
Time
Prefs
Monday 05:10 PM 1995
Current Stream: Eric_Freeman
Date Sunday, Ap 09 11 22 AM 1995 EDT
Summary Re: last significant lifepaper round
From gelernter-david@cs.yale.edu
To Eric_Freeman
Status frozen
Documents
New
Clone
Freeze
Transfer
View Attributes
Yesterday 01:55 PM Image of Watson Hall Image
Yesterday 01:55 PM Voice Mail Msg Audio
Yesterday 11:22 AM Re: last significant lifepaper roun Text
Yesterday 08:32 PM Re: thanks Text Frozen
Yesterday 06:00 PM Re: last significant lifepaper roun Text Frozen
Saturday 03:33 PM Re: last significant lifepaper roun Text Frozen
Saturday 02:31 PM Re: speakers Text Frozen
Saturday 02:01 PM last significant lifepaper round Text Frozen
Saturday 01:58 PM Re: other mac consultant on pcnet Text Frozen
Saturday 01:22 PM thanks Text Frozen
Friday 07:53 PM weights, #1005 Text Frozen
Friday 02:30 PM request Text Frozen
Friday 01:44 PM Re: more notes Text Frozen
Text Frozen
Text Frozen
Text Frozen
Text Frozen
Text Frozen
Text Frozen
Text Frozen
Text Frozen
Friday 10:05 AM Slides Text Frozen
The more interesti
attributes of the la
selected documen
displayed here
The document bu
new documents, c
freeze documents
writable, transfer
and view the docu
(including attache
Documents are displayed in receding chronological order. Each document entry is made up of a configurable set of document attributes. Clicking on an entry displays key attributes above, double-clicking calls the helper application and loads in the document

The STREAM dialog allows the user to connect the viewport to a stream, create and destroy substreams, and perform other stream related functions.

esting
last
ent are

buttons allow users to create
s, clone existing documents,
nts so that they are no longer
fer documents to other users,
ocument attributes in detail
ched agents).

The TIME dialog allows the user to dial to the future and the past.

Through helper applications, based on MIME types, users can view and edit virtually any document type such as text, sound, and images